

МИНОБРНАУКИ РОССИИ

---

Санкт-Петербургский государственный электротехнический  
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

---

Я. А. ТАТЧИНА

**ПРОГРАММНАЯ ЭКОСИСТЕМА  
ДЛЯ РАБОТЫ С ДАННЫМИ**

Учебно-методическое пособие

Санкт-Петербург  
Издательство СПбГЭТУ «ЛЭТИ»  
2020

УДК 004.432(07)

ББК 3 973.233я7

Т12

**Татчина Я. А.**

Т12 Программная экосистема для работы с данными: учеб.-метод. пособие.  
СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2020. 28 с.

ISBN 978-5-7629-2653-9

Рассмотрены основные программные средства и библиотеки языка программирования Python для исследования и анализа данных, приведены практические примеры их использования.

Предназначено для подготовки бакалавров по направлению 09.03.02  
«Информационные системы и технологии».

УДК 004.432(07)

ББК 3 973.233я7

Рецензент: кафедра информационных систем и технологий факультета информатики и прикладной математики ФГБОУ ВО «Санкт-Петербургский государственный экономический университет» (канд. техн. наук, доц. И. Л. Коршунов).

Утверждено  
редакционно-издательским советом университета  
в качестве учебно-методического пособия

ISBN 978-5-7629-2653-9

© СПбГЭТУ «ЛЭТИ», 2020

## **Введение**

Язык программирования Python считается одним из лучших инструментов для анализа, визуализации данных, научных вычислений. Python пригоден для работы с данными благодаря его простоте и большому количеству активно развивающихся открытых библиотек, таких как:

1. NumPy (Numeric Python) – библиотека для работы с однородными данными в виде многомерных массивов и матриц, функционал которой можно сравнить с функционалом MatLab.
2. Pandas – библиотека для работы с неоднородными данными.
3. Matplotlib – пакет для визуализации данных.
4. SciPy – библиотека для выполнения научных и инженерных расчетов.
5. Scikit-Learn – библиотека для машинного обучения.

Это далеко не весь список пакетов для работы с данными. Мы рассмотрим первые три библиотеки.

## 1. ANACONDA И JUPYTER NOTEBOOK

Установка Python и необходимых библиотек не представляет сложности. Сделать это можно несколькими путями: в виде отдельного пакета или дистрибутивом Anaconda.

**Anaconda** – дистрибутив Python, включающий в себя набор библиотек для работы с данными и пакетов для машинного обучения. По состоянию на 2019 год содержит более 1,5 тыс. модулей.

Скачать дистрибутив можно по ссылке: <https://www.anaconda.com/distribution/>.

**Jupyter Notebook** – браузерный графический интерфейс для командной оболочки IPython. Jupyter Notebook позволяет редактировать код в браузере, с подсветкой синтаксиса, автоотступами и автодополнением. Кроме того, в нем можно отображать статические и динамические схемы, графики, вставлять рисунки и даже видеофайлы, работать с языком разметки Markdown и LaTeX (рис. 1.1).

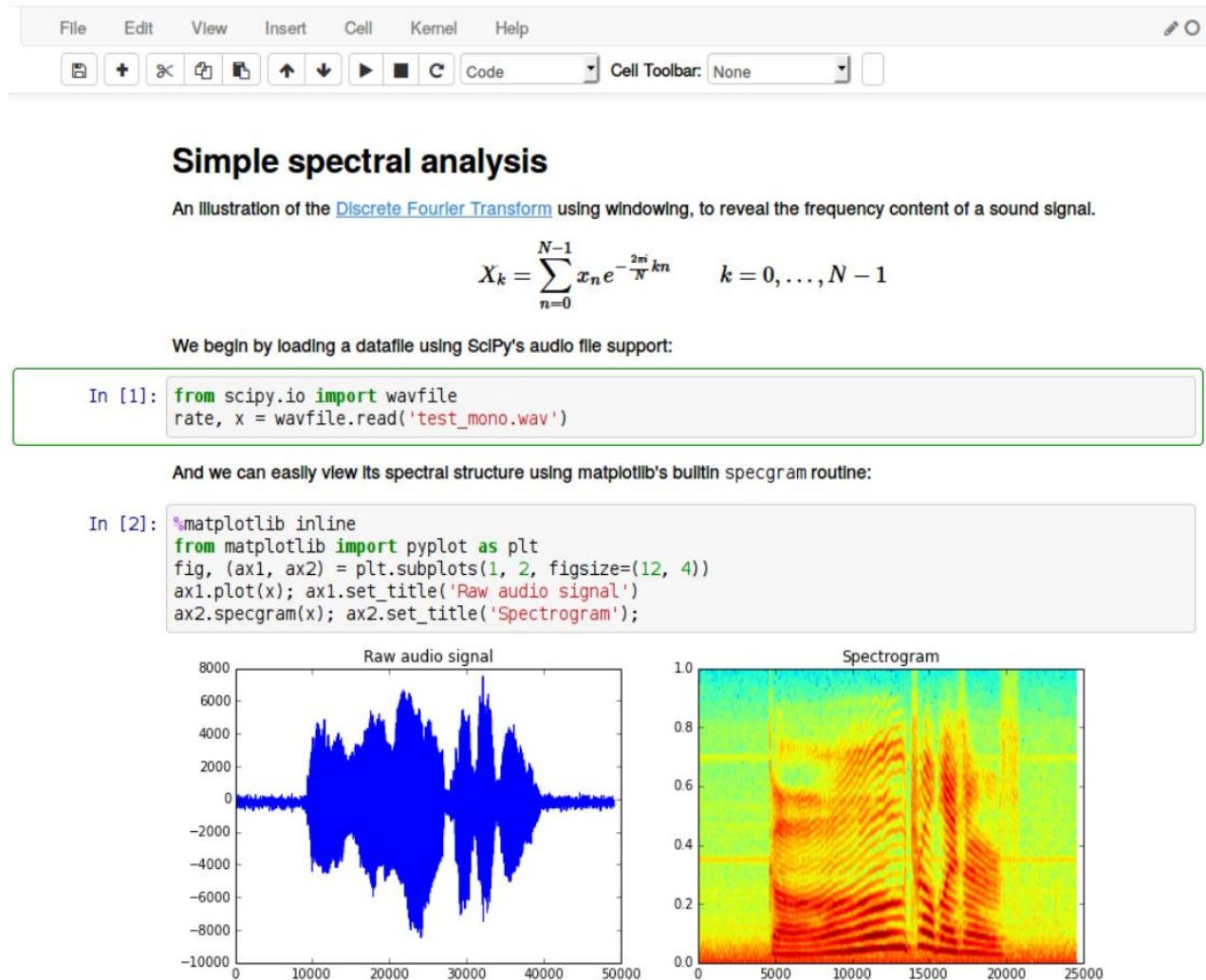
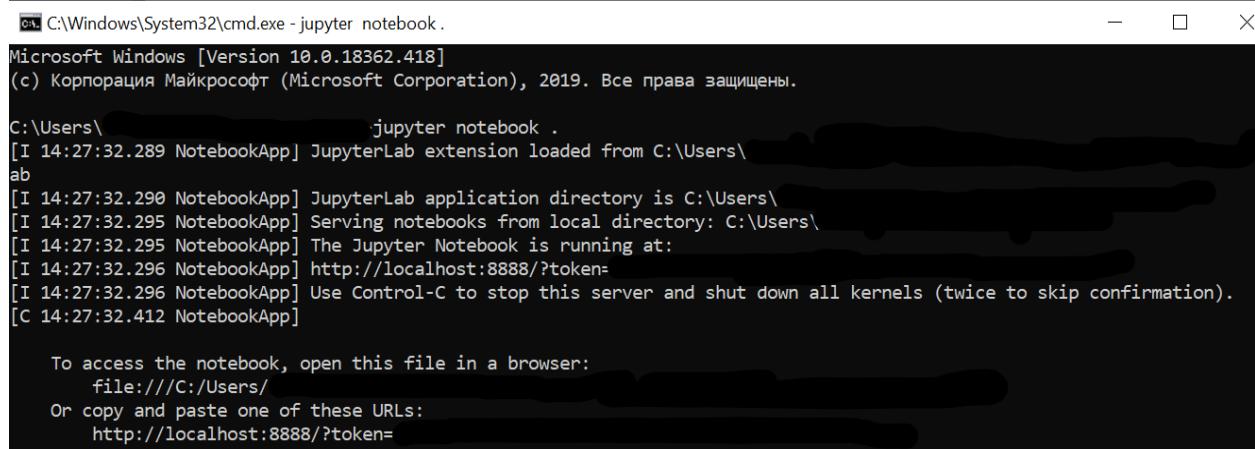


Рис. 1.1

Jupyter Notebook входит в состав Anaconda. Для запуска процесса (называемого ядром (kernel)) необходимо выполнить следующую команду в командной строке:

```
> jupyter notebook
```

Эта команда запустит локальный веб-сервер, видимый браузеру. Она сразу же начнет логировать выполняемые действия. На рис. 1.2 показано, как будет выглядеть журнал.



```
C:\Windows\System32\cmd.exe - jupyter notebook.

Microsoft Windows [Version 10.0.18362.418]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\Users\          jupyter notebook .
[I 14:27:32.289 NotebookApp] JupyterLab extension loaded from C:\Users\ab
[I 14:27:32.290 NotebookApp] JupyterLab application directory is C:\Users\ab
[I 14:27:32.295 NotebookApp] Serving notebooks from local directory: C:\Users\ab
[I 14:27:32.295 NotebookApp] The Jupyter Notebook is running at:
[I 14:27:32.296 NotebookApp] http://localhost:8888/?token=
[I 14:27:32.296 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 14:27:32.412 NotebookApp]

To access the notebook, open this file in a browser:
  file:///C:/Users/ab
Or copy and paste one of these URLs:
  http://localhost:8888/?token=
```

Рис. 1.2

После выполнения этой команды браузер по умолчанию должен автоматически запуститься и перейти по указанному локальному URL.

Рассмотрим несколько примеров работы с ноутбуком на практике.

Запустите Jupyter Notebook, создайте папку для работы, нажав на «New» в правой части экрана, и выберите «Folder» (рис. 1.3).

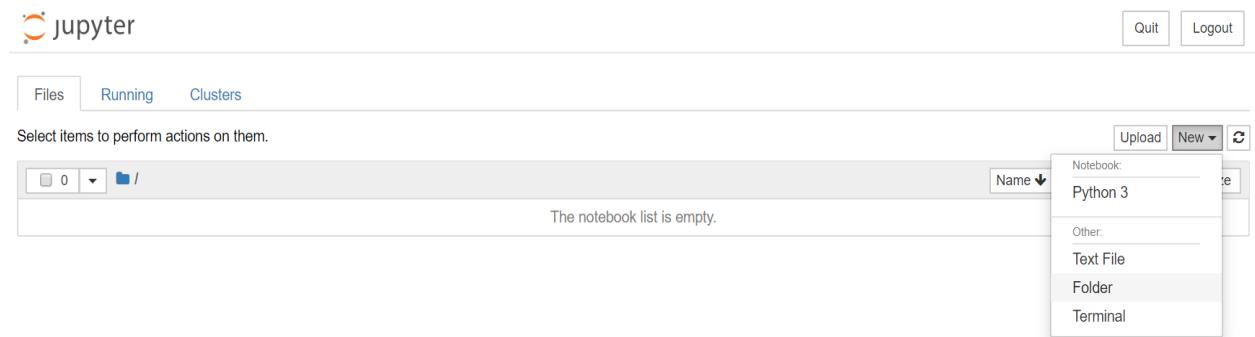


Рис. 1.3

Переименуем папку в «Test» (рис. 1.4).



Рис. 1.4

Зайдем в папку и создадим в ней ноутбук (рис. 1.5).

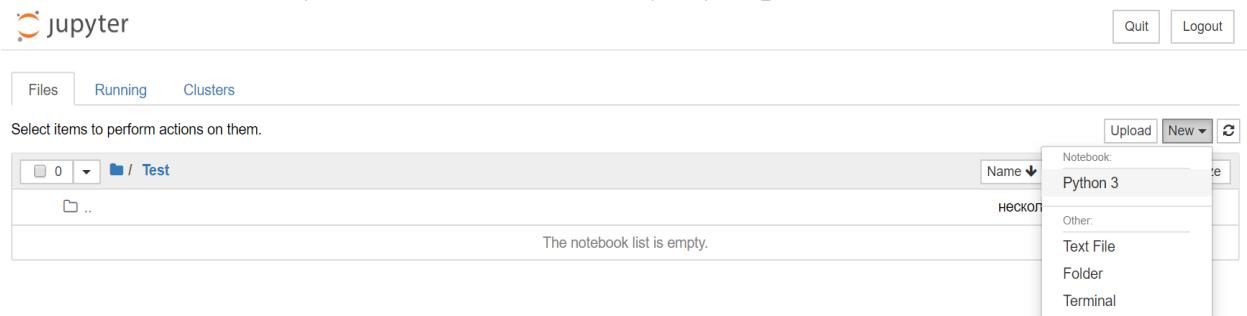


Рис. 1.5

В результате будет создан ноутбук (рис. 1.6).

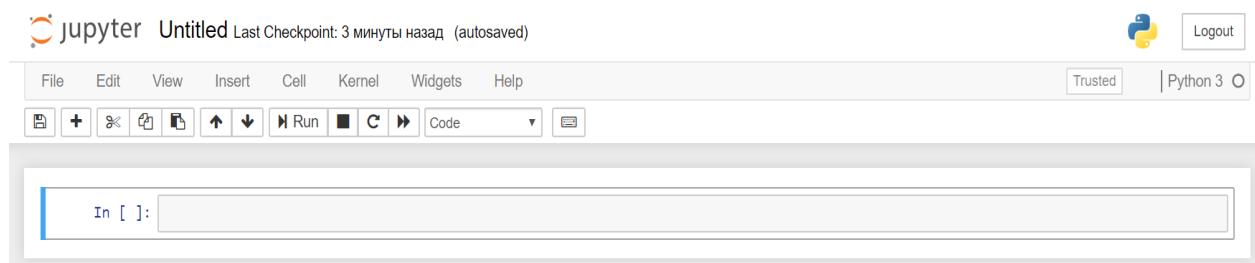


Рис. 1.6

Код или записи в Markdown нужно вводить в ячейки, тип ячейки можно изменить в меню (рис. 1.7).

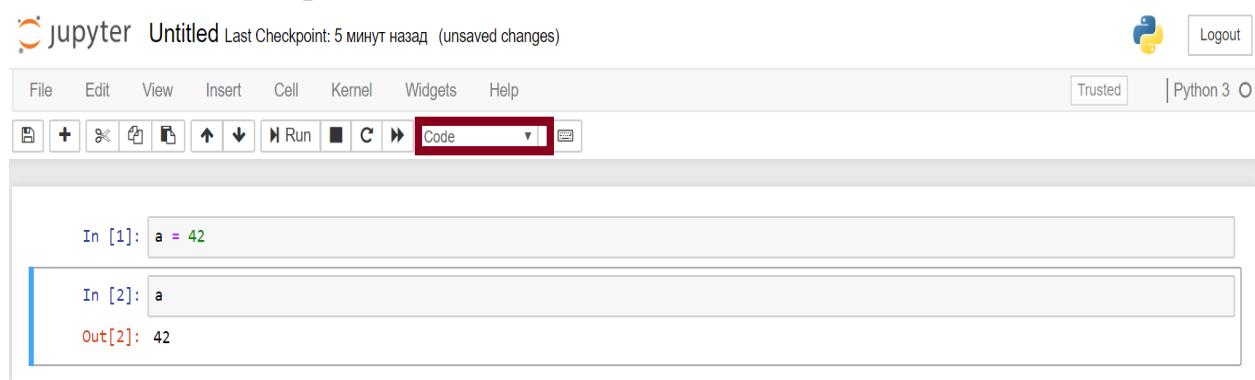


Рис. 1.7

Для выполнения кода нужно нажать Ctrl+Enter или Shift+Enter.

Попробуйте выполнить несколько примеров, показанных на рис. 1.8.

```

1 + 2
3

a = 'Hello'
b = 'world'
c = a + ' ' + b
print(c)

Hello world

my_list = [1, 2, 3, 4, 5]

for i in my_list:
    print(i)

1
2
3
4
5

i = 0

while i < 4:
    print('i = ', i)
    i += 1

i = 0
i = 1
i = 2
i = 3

```

Рис. 1.8

Ноутбук может находиться в двух режимах – режиме правки (Edit mode) и командном (Command mode). Текущий режим отображается на панели меню в правой части (рис. 1.9).

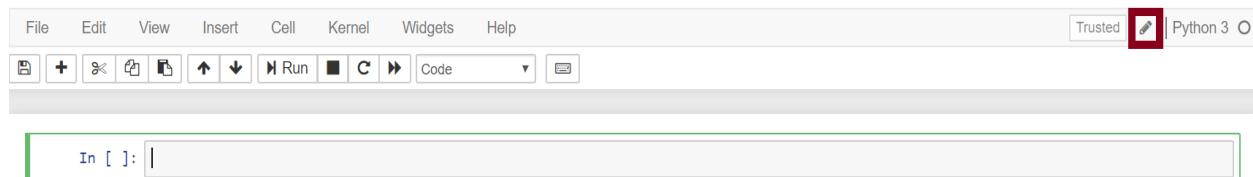


Рис. 1.9

Если программа зависла, то можно прервать ее выполнение, выбрав на панели меню пункт Kernel > Interrupt.

Для добавления новой ячейки используйте Insert > Insert Cell Above и Insert > Insert Cell Below.

Для запуска ячейки используйте команды из меню Cell либо следующие сочетания клавиш:

- Ctrl+Enter – выполнить содержимое ячейки;
- Shift+Enter – выполнить содержимое ячейки и перейти на ячейку ниже;
- Alt+Enter – выполнить содержимое ячейки и вставить новую ячейку ниже.

Важной частью функционала Jupyter Notebook является поддержка magic. Под magic в IPython понимаются дополнительные команды, выполняемые в рамках оболочки, которые облегчают процесс разработки и расширяют ваши возможности. Список доступных магических команд можно получить с помощью команды `%lsmagic` (рис. 1.10).

```
In [7]: %lsmagic
Out[7]: Available line magics:
alias %alias_magic %autoawait %automagic %autosave %bookmark %cd %clear %cls %colors %config %co
nnect_info %copy %ddir %debug %dhist %dirs %doctest_mode %echo %ed %edit %env %gui %hist %history %killbgscripts
%ldir %less %load %load_ext %loadpy %logoff %logon %logstart %logstate %logstop %ls %lsmagic %macro %magic %matpl
otlib %mkdir %more %notebook %page %pastebin %pdb %pdef %pdoc %pfile %pinfo %pinfo2 %pip %popd %pprint %precisio
n %prun %psearch %psource %pushd %pwd %pycat %pylab %qtconsole %quickref %recall %rehashx %reload_ext %ren %rep
%rerun %reset %reset_selective %rmdir %run %save %sc %set_env %store %sx %system %tb %time %timeit %unalias %unl
oad_ext %who %who_ls %whos %xdel %xmode

Available cell magics:
%%! %%HTML %%SVG %%bash %%capture %%cmd %%debug %%file %%html %%javascript %%js %%latex %%markdown %%perl %%prun
%%pypy %%python %%python2 %%python3 %%ruby %%script %%sh %%svg %%sx %%system %%time %%timeit %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

Рис. 1.10

Информацию по командам magic можно найти здесь:  
<https://ipython.org/ipython-doc/3/interactive/magics.html>.

Для быстрого ознакомления с основными командами панельного меню можно воспользоваться User Interface Tour (рис. 1.11).

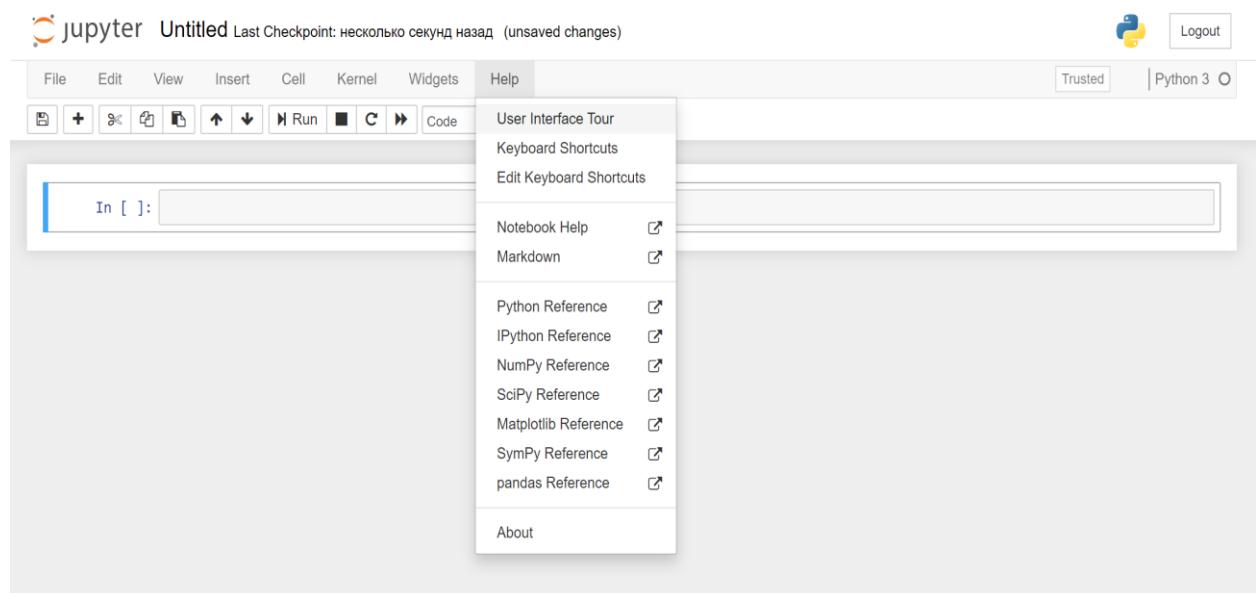


Рис. 1.11

## 2. ВВЕДЕНИЕ В БИБЛИОТЕКУ NUMPY

NumPy (Numerical Python – «числовой Python») – это библиотека языка Python, работающая с большими многомерными массивами и матрицами.

Установить ее можно с дистрибутивом Anaconda или как самостоятельный пакет Python (рис. 2.1).

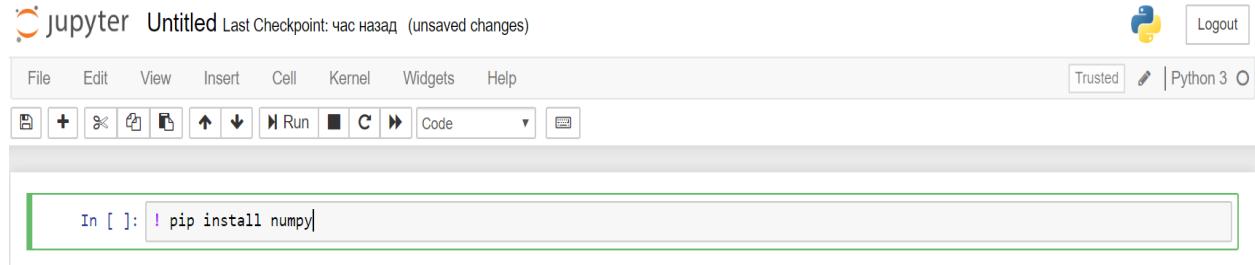


Рис. 2.1

Знак «!» позволяет выполнить код в командной строке.

Существует несколько вариантов импорта. Стандартный метод показан на рис. 2.2.

```
In [1]: import numpy
```

Рис. 2.2

Но по традиции пакет импортируется в качестве псевдонима «пр» (рис. 2.3).

```
In [2]: import numpy as np
```

Рис. 2.3

Главной особенностью NumPy является объект array. Массивы схожи со списками в Python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных. С массивами можно проводить числовые операции с большим объемом информации в несколько раз быстрее и намного эффективнее, чем со списками.

Для создания массивов из списков можно воспользоваться функцией np.array().

Если типы элементов не совпадают, NumPy попытается выполнить повышающее приведение типов (в данном случае целочисленные значения приводятся к числам с плавающей точкой).

Если же необходимо явным образом задать тип данных для итогового массива, можно воспользоваться ключевым словом `dtype`.

Массивы могут быть и многомерными (рис. 2.4).

```
import numpy as np

np.array([1, 2, 3, 4])
array([1, 2, 3, 4])

np.array([3.14, 4, 2, 3])
array([3.14, 4., 2., 3.])

np.array([1, 2, 3, 4], dtype='float32')
array([1., 2., 3., 4.], dtype=float32)

np.array([[1, 2, 3], [4, 5, 6]], float)
array([[1., 2., 3.],
       [4., 5., 6.]])
```

Рис. 2.4

Функция `array()` – не единственная функция для создания массивов. Функция `zeros()` создает массив из нулей, а функция `ones()` – массив из единиц. Обе принимают кортеж с размерами и аргумент `dtype`.

Функция `eye()` создает единичную матрицу (двумерный массив) (рис. 2.5).

```
np.zeros((3, 5))
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]))

np.ones((2, 3, 2))
array([[[1., 1.],
         [1., 1.],
         [1., 1.]],
        [[1., 1.],
         [1., 1.],
         [1., 1.]]])

np.eye(5)
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Рис. 2.5

Для создания последовательностей чисел в NumPy имеется функция `arange()`, аналогичная встроенной в Python `range()`, только здесь она вместо списков возвращает массивы (рис. 2.6).

```
np.arange(10, 30, 5)
```

```
array([10, 15, 20, 25])
```

Рис. 2.6

На рис. 2.7 показаны примеры базовых операций над массивами в NumPy.

```
np.arange(10, 30, 5)
```

```
array([10, 15, 20, 25])
```

```
a = np.array([1, 2, 3], float)  
b = np.array([4, 5, 6], float)
```

```
a + b
```

```
array([5., 7., 9.])
```

```
a - b
```

```
array([-3., -3., -3.])
```

```
a * b
```

```
array([ 4., 10., 18.])
```

```
b / a
```

```
array([4. , 2.5, 2. ])
```

```
a % b
```

```
array([1., 2., 3.])
```

```
b**a
```

```
array([ 4., 25., 216.])
```

Рис. 2.7

Обратите внимание, что функции sum и np.sum неидентичны: функция np.sum() может работать с многомерными массивами.

При умножении двумерных массивов операция происходит поэлементно, что не соответствует умножению матриц. Для таких случаев существуют специальные функции.

Элементы могут быть суммированы или перемножены, можно найти максимальное, минимальное или среднее значения массива (рис. 2.8).

```
a = np.array([5, 4, 1], float)
a.sum()
```

10.0

```
a.mean()
```

3.333333333333335

```
a.min()
```

1.0

```
a.max()
```

5.0

*Рис. 2.8*

Логическое сравнение может быть использовано для поэлементного сравнения массивов одинаковых длин (рис. 2.9).

```
a = np.array([1, 5, 8], float)
b = np.array([0, 0, 2], float)
```

```
a > b
```

array([ True, True, True])

```
a == b
```

array([False, False, False])

```
a <= b
```

array([False, False, False])

*Рис. 2.9*

Полный список операций можно изучить, пройдя по ссылке: <https://docs.scipy.org/doc/numpy/reference/routines.math.html>.

С одномерными массивами можно выполнять операции индексирования, срезов и итераций аналогично операциям со списками в Python (рис. 2.10).

```

a = np.array([1, 2, 3, 56, 84, 93, 2, 55, 98], int)

a[1]
2

a[2 : 6]
array([ 3, 56, 84, 93])

a[::-1]
array([98, 55, 2, 93, 84, 56, 3, 2, 1])

for element in a:
    print(element - 3)

-2
-1
0
53
81
90
-1
52
95

```

Puc. 2.10

```

a = np.array([[0, 1, 2, 3],
             [10, 11, 12, 13],
             [20, 21, 22, 23],
             [30, 31, 32, 33],
             [40, 41, 42, 43]])

a[0, 0]
0

a[3, 3]
33

a[3][3]
33

a[:1]
array([[0, 1, 2, 3]])

a[-1]
array([40, 41, 42, 43])

a[::-1]
array([[40, 41, 42, 43],
       [30, 31, 32, 33],
       [20, 21, 22, 23],
       [10, 11, 12, 13],
       [ 0,  1,  2,  3]])
```

Puc. 2.11

У многомерных массивов индексы передаются в виде последовательности чисел, разделенных запятыми (рис. 2.11).

Итерирование по многомерному массиву показано на рис. 2.12.

```
a = np.array([[0, 1, 2, 3],  
             [10, 11, 12, 13],  
             [20, 21, 22, 23],  
             [30, 31, 32, 33],  
             [40, 41, 42, 43]])
```

```
for row in a:  
    print(row)
```

```
[0 1 2 3]  
[10 11 12 13]  
[20 21 22 23]  
[30 31 32 33]  
[40 41 42 43]
```

Рис. 2.12

Полная документация доступна по адресу:  
<https://numpy.org/devdocs/user/quickstart.html>.

### 3. АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ PANDAS

Pandas – пакет Python для обработки и анализа данных, надстройка над библиотекой NumPy. Данные в Pandas хранятся в виде Series и DataFrame.

Series – это проиндексированный одномерный массив значений.

DataFrame – это проиндексированный многомерный массив значений, каждый столбец DataFrame является Series.

Pandas входит в состав дистрибутива Anaconda.

На рис. 3.1 показана отдельная установка библиотеки Pandas.

```
In [1]: ! pip install pandas
```

Рис. 3.1

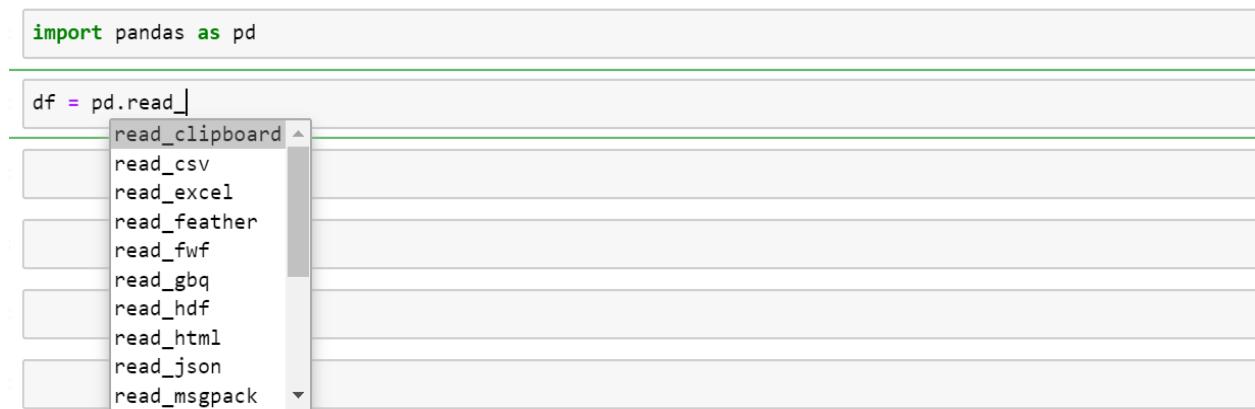
Аналогично тому, как импортировали пакет NumPy под псевдонимом «np», пакет Pandas импортируется под псевдонимом «pd» (рис. 3.2).

```
In [2]: import pandas as pd
```

Рис. 3.2

Рассмотрим основные свойства и функции работы библиотеки Pandas на примере данных об Олимпийских играх с 1896 по 2016 г. Скачать данные можно по ссылке: <https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results/download>.

Pandas умеет считывать данные разных типов. Для чтения используется функция pd.read\_тип\_файла (Tab позволяет использовать autocomplete) (рис. 3.3).



```
import pandas as pd

df = pd.read_
```

read\_clipboard

- read\_csv
- read\_excel
- read\_feather
- read\_fwf
- read\_gbq
- read\_hdf
- read\_html
- read\_json
- read\_msgpack

Рис. 3.3

Импортируем библиотеку, считаем данные и выведем первые 5 строк (рис. 3.4).

```
import pandas as pd

df = pd.read_csv('athlete_events.csv')
df.head()
```

	Name	Sex	Team	Year	Season	City	Sport	Medal
0	A Dijiang	M	China	1992	Summer	Barcelona	Basketball	NaN
1	A Lamusi	M	China	2012	Summer	London	Judo	NaN
2	Gunnar Nielsen Aaby	M	Denmark	1920	Summer	Antwerpen	Football	NaN
3	Edgar Lindenau Aabye	M	Denmark/Sweden	1900	Summer	Paris	Tug-Of-War	Gold
4	Christine Jacoba Aafink	F	Netherlands	1988	Winter	Calgary	Speed Skating	NaN

Рис. 3.4

Как можно заметить, DataFrame представляет собой таблицу, где в качестве столбцов выступает объект Series. Для обращения к Series используется синтаксис, показанный на рис. 3.5.

```
df['Games']
```

```
0      1992 Summer
1      2012 Summer
2      1920 Summer
3      1900 Summer
4      1988 Winter
5      1988 Winter
6      1992 Winter
7      1992 Winter
8      1994 Winter
9      1994 Winter
10     1992 Winter
```

Рис. 3.5

Объект Series служит адаптером как для последовательности значений, так и для последовательности индексов, к которым можно получить доступ посредством атрибутов values и index (рис. 3.6).

```
df['Games'].index
```

```
RangeIndex(start=0, stop=271116, step=1)
```

```
df['Games'].values
```

```
array(['1992 Summer', '2012 Summer', '1920 Summer', ..., '2014 Winter',
       '1998 Winter', '2002 Winter'], dtype=object)
```

Рис. 3.6

Аналогично массивам библиотеки NumPy к данным можно обращаться по соответствующему им индексу (рис. 3.7).

```
df['Games'][0]
```

```
'1992 Summer'
```

```
df['Games'][0:2]
```

```
0    1992 Summer
1    2012 Summer
Name: Games, dtype: object
```

Рис. 3.7

Основное различие между Series и одномерным массивом NumPy – индекс. В то время как индекс массива NumPy, используемый для доступа к значениям, целочисленный и описывается неявно, индекс объекта Series библиотеки Pandas описывается явно и связывается со значениями.

Аналогично объекту Series у объекта DataFrame имеются атрибут index, обеспечивающий доступ к меткам индекса, и атрибут columns (рис. 3.8).

```
df.index  
RangeIndex(start=0, stop=271116, step=1)
```

```
df.columns  
Index(['ID', 'Name', 'Sex', 'Age', 'Height', 'Weight', 'Team', 'NOC', 'Games',  
       'Year', 'Season', 'City', 'Sport', 'Event', 'Medal'],  
      dtype='object')
```

Рис. 3.8

Таким образом, объект DataFrame можно рассматривать как обобщение двумерного массива NumPy, где как у строк, так и у столбцов есть обобщенные индексы для доступа к данным.

Подробная информация об объекте DataFrame доступна по ссылке: <http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>.

Доступ к строкам по индексу возможен несколькими способами (рис. 3.9):

```
df.loc[0]  
  
ID                      1  
Name                   A Dijiang  
Sex                      M  
Age                     24  
Height                  180  
Weight                   80  
Team                     China  
NOC                      CHN  
Games                1992 Summer  
Year                     1992  
Season                   Summer  
City                      Barcelona  
Sport                     Basketball  
Event        Basketball Men's Basketball  
Medal                      NaN  
Name: 0, dtype: object
```

```
df.loc[[0, 1], 'Sport']  
  
0    Basketball  
1      Judo  
Name: Sport, dtype: object
```

```
df.iloc[0 : 2, 1 : 6] # Стока с 1 по 2, столбцы со 2 по 6
```

	Name	Sex	Age	Height	Weight
0	A Dijiang	M	24.0	180.0	80.0
1	A Lamusi	M	23.0	170.0	60.0

Рис. 3.9

- loc – используется для доступа по строковой метке;
  - iloc – используется для доступа по числовому значению (начиная от 0).
- Фильтровать DataFrame можно с помощью булевых массивов. Логические операторы «И» («AND»), «ИЛИ» («OR»), «НЕ» («NOT») в Pandas определяются знаками «&», «|», «~» соответственно.

Найдем всех мужчин из Лондона, участвовавших в Олимпийских играх после 2003 г. (рис. 3.10).

```
df[(df['Sex'] == 'M') & (df['City'] == 'London') & (df['Year'] > 2003)]
```

		Name	Sex	Team	Year	Season	City	Sport	Medal
1		A Lamusi	M	China	2012	Summer	London	Judo	NaN
98	Jamale (Djamel-) Aarrass (Ahrass-)		M	France	2012	Summer	London	Athletics	NaN
134		Abdelhak Aataknii	M	Morocco	2012	Summer	London	Boxing	NaN
174		Luc Abalo	M	France	2012	Summer	London	Handball	Gold
197		Emanuele Abate	M	Italy	2012	Summer	London	Athletics	NaN
...		...	...	...	...	...	...	...	...
271004	Krzysztof Maciej Zwarycz		M	Poland	2012	Summer	London	Weightlifting	NaN
271020		Dniel Zwickl	M	Hungary	2012	Summer	London	Table Tennis	NaN
271023		Marc Zwiebler	M	Germany	2012	Summer	London	Badminton	NaN
271090		Dominik ycki	M	Poland	2012	Summer	London	Sailing	NaN
271091	ukasz Tomasz ygado		M	Poland	2012	Summer	London	Volleyball	NaN

Рис. 3.10

```
df[(df['Team'] == 'Russia') & ((df['Medal'] == 'Gold') | (df['Medal'] == 'Silver'))]
```

		Name	Sex	Team	Year	Season		City	Sport	Medal
163	Mariya Vasilyevna Abakumova (-Tarabina)		F	Russia	2008	Summer		Beijing	Athletics	Silver
195	Tamila Rashidovna Abasova		F	Russia	2004	Summer		Athina	Cycling	Silver
790	Denis Mikhaylovich Ablyazin		M	Russia	2012	Summer		London	Gymnastics	Silver
794	Denis Mikhaylovich Ablyazin		M	Russia	2016	Summer	Rio de Janeiro		Gymnastics	Silver
796	Denis Mikhaylovich Ablyazin		M	Russia	2016	Summer	Rio de Janeiro		Gymnastics	Silver
...		...	...	...	...	...	...	...	...	...
269789	Irina Aleksandrovna Zilber		F	Russia	2000	Summer		Sydney	Rhythmic Gymnastics	Gold
270009	Irek Khaydarovich Zinnurov		M	Russia	2000	Summer		Sydney	Water Polo	Silver
270934	Nataliya Vladimirovna Zuyeva		F	Russia	2008	Summer		Beijing	Rhythmic Gymnastics	Gold
270939	Anastasiya Valeryevna Zuyeva-Fesikova		F	Russia	2012	Summer		London	Swimming	Silver
271103	Olesya Nikolayevna Zykina		F	Russia	2004	Summer		Athina	Athletics	Silver

Рис. 3.11

Другой пример: найдем все данные о команде из России по выигрышам золотой или серебряной медали (рис. 3.11).

Добавим новый столбец: рассчитаем индекс массы тела для каждого участника по формуле ИМТ = вес (кг) / рост (м) и округлим до целой части (рис. 3.12).

```
df['BMI'] = round(df['Weight'] / (df['Height'] / 100))
```

```
df
```

	Name	Sex	Team	Year	Weight	Height	Sport	Medal	BMI
0	A Dijiang	M	China	1992	80.0	180.0	Basketball	NaN	44.0
1	A Lamusi	M	China	2012	60.0	170.0	Judo	NaN	35.0
2	Gunnar Nielsen Aaby	M	Denmark	1920	NaN	NaN	Football	NaN	NaN
3	Edgar Lindenau Aabye	M	Denmark/Sweden	1900	NaN	NaN	Tug-Of-War	Gold	NaN
4	Christine Jacoba Aaftink	F	Netherlands	1988	82.0	185.0	Speed Skating	NaN	44.0
...	...	...	...	...	...	...	...	...	...
271111	Andrzej ya	M	Poland-1	1976	89.0	179.0	Luge	NaN	50.0
271112	Piotr ya	M	Poland	2014	59.0	176.0	Ski Jumping	NaN	34.0
271113	Piotr ya	M	Poland	2014	59.0	176.0	Ski Jumping	NaN	34.0
271114	Tomasz Ireneusz ya	M	Poland	1998	96.0	185.0	Bobsleigh	NaN	52.0
271115	Tomasz Ireneusz ya	M	Poland	2002	96.0	185.0	Bobsleigh	NaN	52.0

Рис. 3.12

Как удалить столбец, показано на рис. 3.13.

```
df.drop(['Name'], axis=1, inplace=True)
```

```
df.head()
```

	Sex	Team	Year	Weight	Height	Sport	Medal	BMI
0	M	China	1992	80.0	180.0	Basketball	NaN	44.0
1	M	China	2012	60.0	170.0	Judo	NaN	35.0
2	M	Denmark	1920	NaN	NaN	Football	NaN	NaN
3	M	Denmark/Sweden	1900	NaN	NaN	Tug-Of-War	Gold	NaN
4	F	Netherlands	1988	82.0	185.0	Speed Skating	NaN	44.0

Рис. 3.13

Метод `drop()` принимает значения: индекс\ столбец, который необходимо удалить; `axis = 1` или `0`, если `1` – удалить столбец, `0` – строку; `inplace = True` или `False` – изменить данные или нет.

Важная часть анализа больших данных – их эффективное обобщение: вычисление сводных показателей, например `sum()`, `mean()`, `median()`, `min()` и `max()` (рис. 3.14).

```
df['Weight'].min()
```

```
25.0
```

```
df['Weight'].max()
```

```
214.0
```

```
df['Age'].mean()
```

```
25.556898357297374
```

Рис. 3.14

В таблице перечислены основные встроенные агрегирующие методы библиотеки Pandas.

#### Агрегирующие методы библиотеки Pandas

Агрегирующая функция	Описание
<code>count()</code>	Общее количество элементов
<code>first()</code> , <code>last()</code>	Первый и последний элементы
<code>mean()</code> , <code>median()</code>	Среднее значение и медиана
<code>min()</code> , <code>max()</code>	Минимум и максимум
<code>std()</code> , <code>var()</code>	Стандартное отклонение и дисперсия
<code>mad()</code>	Среднее абсолютное отклонение
<code>prod()</code>	Произведение всех элементов
<code>sum()</code>	Сумма всех элементов

У DataFrame есть полезный метод `describe()`, который позволяет вычислить некоторые математические свойства числовых данных (рис. 3.15).

Группировка – один из самых часто используемых методов при анализе данных. В Pandas за группировку отвечает метод `groupby()`, который необходимо выполнять с агрегирующей функцией.

В качестве примера получим данные о том, сколько всего медалей выиграли женщины и мужчины в Олимпийских играх с 1896 по 2016 г. (рис. 3.16).

```
df.describe()
```

	ID	Age	Height	Weight	Year
count	271116.000000	261642.000000	210945.000000	208241.000000	271116.000000
mean	68248.954396	25.556898	175.338970	70.702393	1978.378480
std	39022.286345	6.393561	10.518462	14.348020	29.877632
min	1.000000	10.000000	127.000000	25.000000	1896.000000
25%	34643.000000	21.000000	168.000000	60.000000	1960.000000
50%	68205.000000	24.000000	175.000000	70.000000	1988.000000
75%	102097.250000	28.000000	183.000000	79.000000	2002.000000
max	135571.000000	97.000000	226.000000	214.000000	2016.000000

Рис. 3.15

```
df.groupby('Sex')['Medal'].count()
```

```
Sex
F    11253
M    28530
Name: Medal, dtype: int64
```

Рис. 3.16

Другой пример: определим ТОП 15 стран по количеству медалей и отсортируем список по убыванию (рис. 3.17).

```
df.groupby('Team')['Medal'].count().sort_values(ascending=False).head(15)
```

```
Team
United States    5219
Soviet Union     2451
Germany          1984
Great Britain    1673
France            1550
Italy              1527
Sweden            1434
Australia         1306
Canada            1243
Hungary           1127
Russia             1110
Netherlands       988
East Germany      941
Japan              911
Norway             910
Name: Medal, dtype: int64
```

Рис. 3.17

Вы могли обратить внимание, что в таблице имеются значения NaN (Not-a-Number) – так заполняются отсутствующие данные. Существует множество эвристик, как обрабатывать эти значения: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/missing\\_data.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html).

Посчитаем, сколько пропущенных значений у ['Medal'] в DataFrame (рис. 3.18).

```
df['Medal'].isna().sum()  
231333
```

Рис. 3.18

Также можно посмотреть на уникальные значения Series (рис. 3.19).

```
df['Medal'].unique()  
array([nan, 'Gold', 'Bronze', 'Silver'], dtype=object)
```

Рис. 3.19

Сохранить DataFrame можно в файлы почти любого типа (рис. 3.20).

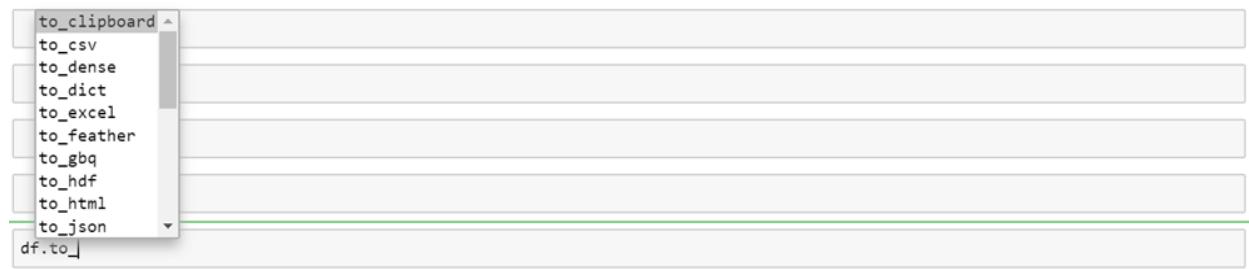


Рис. 3.20

Сохраним измененные данные в csv-файл (рис. 3.21).

```
df.to_csv('Changed_file.csv')
```

Рис. 3.21

С полной документацией Pandas можно ознакомиться на сайте: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/index.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html).

## 4. ВИЗУАЛИЗАЦИЯ С ПОМОЩЬЮ БИБЛИОТЕКИ MATPLOTLIB

Matplotlib – библиотека Python, предназначенная для визуализации данных.

На рис. 4.1 показаны установка и импорт библиотеки.

```
! pip install matplotlib  
  
import matplotlib.pyplot as plt  
%matplotlib inline
```

Рис. 4.1

Команда %matplotlib inline в ячейке Jupyter Notebook позволяет визуализировать данные прямо в окне браузера.

Рассмотрим примеры построения графиков на данных, загруженных в предыдущей работе с Pandas.

Гистограмма распределения возраста спортсменов показана на рис. 4.2.

```
plt.figure(figsize=(10, 5)) # параметры
plt.hist(df['Age'], bins=30); # гистограмма распределения возраста спортсменов
plt.grid(True) # вспомогательная сетка
plt.title('Age distribution');
```

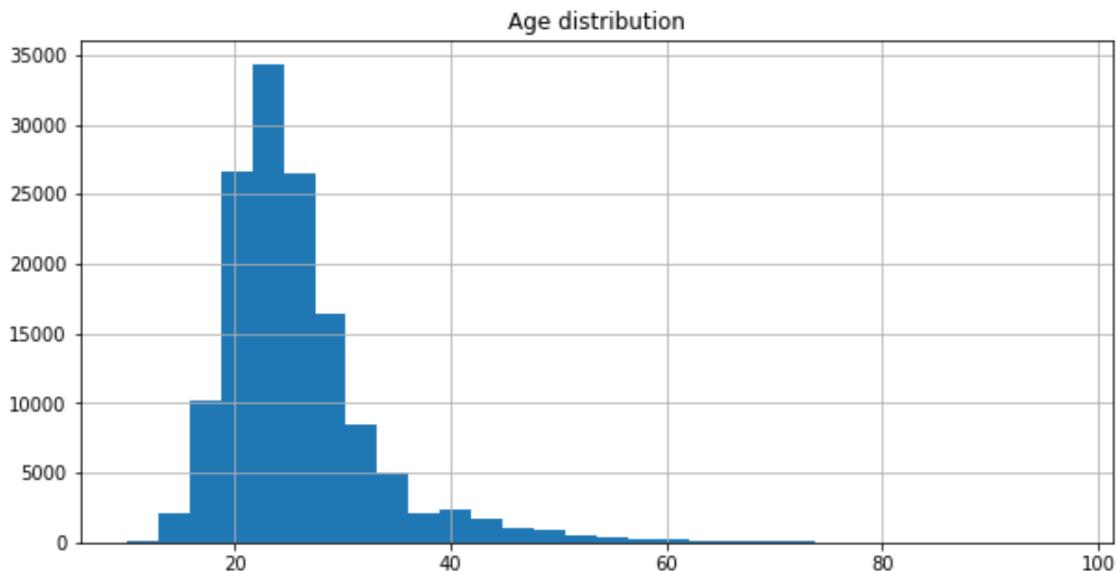


Рис. 4.2

Диаграмма рассеяния (зависимость веса от роста спортсменов) приведена на рис. 4.3.

```
plt.figure(figsize=(10, 5))
plt.scatter(df['Height'], df['Weight']);
plt.xlabel('Height');
plt.ylabel('Weight');
plt.title('Dependence of height on weight');
```

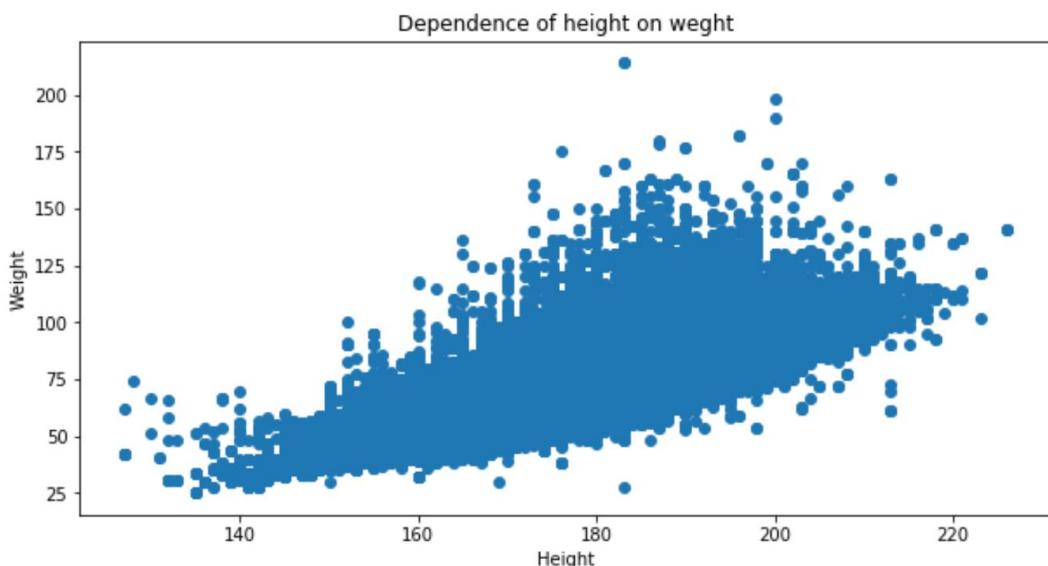


Рис. 4.3

График зависимости количества медалей от года проведения Олимпийских игр показан на рис. 4.4.

```
plt.figure(figsize=(10, 5))
plt.plot(df.groupby('Year')[ 'Medal'].count());
plt.xlabel('Year');
plt.ylabel('Number of medals');
plt.title('Dependence of the number of medals on the year');
```

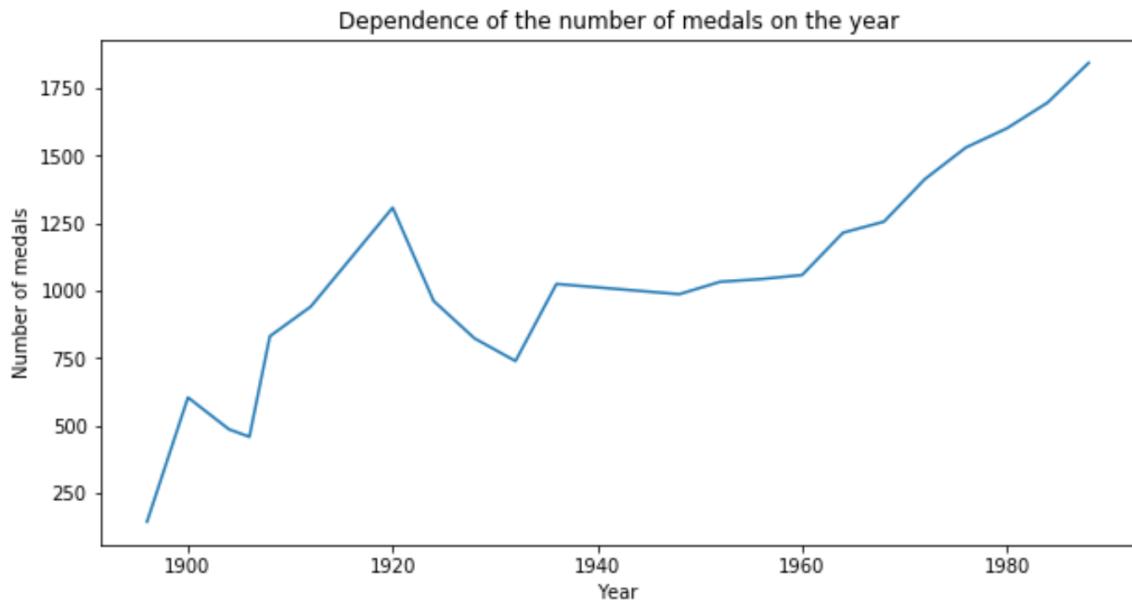


Рис. 4.4

Подробная информация представлена в документации библиотеки Matplotlib по ссылке: <https://matplotlib.org/>.

## **ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ**

1. Какие библиотеки для анализа данных вы знаете?
2. Что такое Jupyter Notebook?
3. Какими способами можно установить библиотеки Python для анализа данных?
4. В чем преимущество дистрибутива Anaconda?
5. Какие существуют способы создания массивов в NumPy?
6. Как импортируется библиотека Pandas?
7. В чем отличие Series от массива NumPy?
8. Что такое DataFrame?
9. Какчитываются данные в Pandas?
10. В чем отличия loc от iloc?
11. Как удалить данные из DataFrame?
12. Как записать данные в csv-файл?
13. Как строится диаграмма рассеяния в Matplotlib?
14. Что выполняет команда %matplotlib inline в Jupyter Notebook?

## **СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ**

Брюс П., Брюс Э. Практическая статистика для специалистов Data Science / пер. с англ. СПб.: БХВ-Петербург, 2018. 304 с.

Вандер Плас. Дж. Python для сложных задач: наука о данных и машинное обучение. СПб.: Питер, 2018. 576 с.

Грас Дж. Наука о данных с нуля / пер. с англ. СПб.: БХВ-Петербург, 2019. 336 с.

## **ОГЛАВЛЕНИЕ**

Введение.....	3
1. ANACONDA И JUPYTER NOTEBOOK .....	4
2. ВВЕДЕНИЕ В БИБЛИОТЕКУ NUMPY .....	9
3. АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ PANDAS .....	14
4. ВИЗУАЛИЗАЦИЯ С ПОМОЩЬЮ БИБЛИОТЕКИ MATPLOTLIB .....	22
ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ .....	25
СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ .....	26

Татчина Яна Александровна

**Программная экосистема для работы с данными**

Учебно-методическое пособие

Редактор Н. В. Кузнецова

---

Подписано в печать 30.06.20. Формат 60×84 1/16.

Бумага офсетная. Печать цифровая. Печ. л. 1,75.

Гарнитура «Times New Roman». Тираж 69 экз. Заказ .

---

Издательство СПбГЭТУ «ЛЭТИ»  
197376, С.-Петербург, ул. Проф. Попова, 5